Stefan Tilkov | innoQ Deutschland GmbH | stefan.tilkov@innoq.com

Metaprogramming Ruby: Writing Code that Writes Code

Contents

Metaprogramming Motivation

1. Languages are Not Equal

2. Repetition Kills Productivity

3. Language Influences Thought

4. Languages Must Evolve

1. Languages are Not Equal

Machine Code
Assembler
C
C++
Java
Python
Ruby
Scheme/Lisp

- All Turing-complete: every task doable in all of them
- Big differences in runtime behavior (speed, efficiency)
- Even bigger differences in development support

Any sufficiently complicated C or Fortran program contains an ad-hoc, informally-specified, bug-ridden, slow implementation of half of CommonLisp.

Philip Greenspun's Tenth Rule of Programming
http://philip.greenspun.com/research/

2. Repetition Kills Productivity

- Repetition equals redundancy
- Manual pattern execution introduces errors ...
- ... and spoils the fun
- Changes become harder, quality decreases
- The less powerful your language, the more you repeat yourself

3. Language Influences Thought

- You only apply patterns and concepts that you know of
- A programming language’s capabilities influence the way you express a solution
- Anything out of the ordinary seems “weird”

“We cut nature up, organize it into concepts, and ascribe significances as we do, largely because we are parties to an agreement to organize it in this way — an agreement that holds throughout our speech community and is codified in the patterns of our language.

Whorf, Benjamin (John Carroll, Editor) (1956). Language, Thought, and Reality: Selected Writings of Benjamin Lee Whorf. MIT Press.
“Sapir-Whorf Hypothesis” (note: now disputed);
see also http://en.wikipedia.org/wiki/Sapir-Whorf_hypothesis

Blub falls right in the middle of the abstractness continuum... As long as our hypothetical Blub programmer is looking down the power continuum, he knows he's looking down. Languages less powerful than Blub are obviously less powerful, because they're missing some feature he's used to.

But when our hypothetical Blub programmer looks in the other direction, up the power continuum, he doesn't realize he's looking up. What he sees are merely weird languages... Blub is good enough for him, because he thinks in Blub.

Paul Graham, “Beating the Averages”
http://www.paulgraham.com/avg.html

4. Languages Must Evolve

- *General* purpose programming languages can cover *general* cases
- Abstractions match *every* domain
- Key idea of DSLs: A language suitable to the *specific* problem domain
- A growable language enables definition of new constructs that look and feel *as if they were part of the language*

1. Languages are Not Equal
2. Repetition is Productivity's Natural Enemy
3. Language Influences Thought
4. Languages Must Evolve

The World's Quickest Ruby Intro

Ruby history

- First release: 1995
- Brainchild of Yukihiro "Matz" Matsumoto
- Based on Perl, Smalltalk, Lisp
- Popularized (a little) by the Pragmatic Programmers (Dave Thomas & Andy Hunt) ~2001
- Popularized (a lot) by Ruby on Rails ~2004

Basic Language Characteristics

- Purely object-oriented
- Type system:
  - dynamic
  - strong
  - implicit
- Interpreted (mostly ...)
- *Very* dynamic
- Strong metaprogramming support

Basics

```
puts "Hello World"
num = 5
if num > 4 then
  puts "num > 4"
elsif num <= 4 then
  puts "num <= 4"
else
  puts "WTF?"
end
puts "num is 5" unless num != 5
```

Loops

```
for i in (1..10) do
  puts i
end
i = 0
while i < 10 do
  puts i
  i += 1
end
```

Comments and Heredocs

```
# One line comment
=begin
A comment spanning multiple lines
=end

text << <<-EOT
 A really long text, simply written as is in its literal form.
 Don't worry about any escaping.
EOT
```

Iteration & Blocks

```
# don't do this
array = ["alpha", "beta", "gamma"]
for i in 0..2 do
  puts array[i]
end
# much better
array.each { | elem | puts elem }

1.upto(10) { | x | puts x }
1.upto(10) { | x | puts "Count: #{x}" }
1.upto(10) do | x |
  puts "Count: #{x}"
end
```

Enumerable

```
slist = %w(alpha beta gamma delta epsilon) # => ["alpha", "beta", "gamma", "delta", "epsilon"
slist.reverse # => ["epsilon", "delta", "gamma", "beta", "alpha"]
slist.map { |elem| elem.reverse } # => ["ahpla", "ateb", "ammag", "atled", "nolispe"]
slist.inject("") { |mush, elem| mush << elem } # => "alphabetagammadeltaepsilon"
slist # => ["alpha", "beta", "gamma", "delta", "epsilon"]
nlist = [1, 2, 3, 4, 5]
nlist.inject { |sum, elem | sum += elem} # => 15
slist # => ["alpha", "beta", "gamma", "delta", "epsilon"]
slist.find { |elem| elem.length > 4} # => "alpha"
slist.collect { |elem| elem.length > 4} # => [true, false, true, true, true]
slist.select { |elem| elem.length > 4} # => ["alpha", "gamma", "delta", "epsilon"]
nlist.max # => 5
slist.max { |a, b| a.length <=> b.length } # => "epsilon"
```

Hashes

```
hash = { "one" => '1', "two" => '2', "three" => '3'}
puts hash["one"]
table = { "p1" => { "last" => "Schulze", "first" => "Hans"},
          "p2" => { "last" => "Meier", "first" => "Klaus"}
        }
puts table["p1"]
puts table["p1"]["first"]
require 'pp'
pp table
pp table["p1"]
```

A Little Bit of Java ...

```java
package com.example;

import java.util.List;
import java.util.Arrays;
import java.util.Collections;
import java.util.Comparator;

public class SortList {
    public static void main(String[] args) {
        List<String> list = Arrays.asList("Shamelessly", "Stolen",
                                          "From", "Ola", "Bini");
         Collections.sort(list, new Comparator<String>() {
            public int compare(String first, String second) {
                return first.length() - second.length();
            }
        });

        String sep = "";
        for (String name : list) {
            System.out.print(sep);
            System.out.print(name);
            sep = ", ";
        }

        System.out.println();
    }
}
```

... vs. Ruby

```ruby
list = ["Shamelessly", "Stolen", "From", "Ola", "Bini"]
puts list.sort_by(&:length).join(', ')
```

http://youtube.com/watch?v=PfnP-8XbJao

Methods

```
def mymethod(a, b, c)
  puts "a = #{a}, b = #{b}, c=#{c}"
end
mymethod(1, 2, 3)
mymethod 1, 2, 3
```

Classes

```ruby
class Person
  @@people_count = 0
  def initialize(first, last)
    @first = first
    @last = last
    @id = @@people_count
    @@people_count += 1
  end

  def to_s
    "#{@last}, #{@first}"
  end
end
p = Person.new("John", "Doe")
puts p
```

Inheritance

```
class Friend < Person
  def initialize(first, last, nick)
    super(first, last)
    @nick = nick
  end

  def drink
    puts "Cheers from #{@nick}"
  end

  def to_s
    "#{super.to_s}, a.k.a. #{@nick}"
  end
end
f = Friend.new("Jack", "Daniels", "Buddy")
puts f
f.drink
```

Modules

```
module M1
  def self.module_method(s)
    puts "Module method: #{s}"
  end

  def mixin
    puts "Value of a: #{@a}"
  end
end
```

```
class X
  include M1
  def initialize
    @a = 4711
  end
end
```

```
M1.module_method("hello")
```

```
x = X.new
x.mixin
```

“Getters” and “Setters”

```
class AttributeHolder
  def name=(n)
    @name = n
  end

  def name
    @name
  end
end


ah = AttributeHolder.new
ah.name = "AH Test"
puts ah.name
```

“Getters” and “Setters” (2)

```
class AttributeHolder2
  def name=(n)
    @name = n
  end

  def name
    @name
  end
  def first_name=(n)
    @first_name = n
  end

  def first_name
    @first_name
  end
end
```

```
ah = AttributeHolder2.new
ah.name = "AH Test"
ah.first_name = "AH First"
puts ah.name, ah.first_name
```

attribute_accessor

```ruby
class AttributeHolder3
  attr_accessor :name, :first_name
end
```

```ruby
ah = AttributeHolder3.new
ah.name = "AH Test"
ah.first_name = "AH First"
puts ah.name, ah.first_name
```

Ruby
vs.
Groovy, Python, Jython, Perl, ...

Ruby vs. Groovy, Python, Jython, Perl, ...

- Perl
  - most extensive set of libraries
  - “write only” reputation, whether merited or not
  - weak (or weird?) OO features
  - very fast
- Python
  - great community
  - extensive libraries
  - extremely fast

Ruby vs. Groovy, Python, Jython, Perl, … (2)

- Jython
  - JVM version of Python
  - neglected for a long time
  - now supported by Sun
- Groovy
  - designed for the JVM
  - supposedly easier for Java developers

Java vs. Groovy vs. Ruby

```
private static < T > List< List< T >> subn(int n, List< T > li) {
  List< List< T > > ret = new ArrayList< List< T >>();
  if (n  0) {
      ret.add(new ArrayList< T >());
      return ret;
  }
  if (li.isEmpty()) {
      return ret;
  }
  T x = li.get(0);
  List< T> xs = li.subList(1, li.size());
  for (List< T> sub : subn(n-1, xs)) {
      sub.add(0, x);
      ret.add(sub);
  }
  ret.addAll(subn(n, xs));
  return ret;

}
```

http://www.glenstampoultzis.net/blog/?p=61

Groovy

```
def subn(n, list) {
    if (n  0) return [[]];
    if (list.isEmpty()) return [];

    def remainder = list.subList(1, list.size());
    return subn(n-1, remainder).collect { [list[ 0 ]] + it } + subn(n, remainder);

}
```

(J)Ruby

```
def subn(n, list)
    return [[]] if n == 0
    return [] if list.empty?

    remainder = list[1..-1]
    subn(n-1, remainder).collect {|it| [list[0]] + it } + subn(n, remainder)
end
```

http://headius.blogspot.com/2008/04/converting-groovy-to-ruby.html

Calling Java from JRuby:

```
require 'java'

JFrame = javax.swing.JFrame
JLabel = javax.swing.JLabel

frame = JFrame.new
frame.getContentPane().add(JLabel.new("Hello from JRuby!"))
frame.pack
frame.setVisible true
frame.show
```

Calling JRuby from Java: JSR 223

```java
import javax.script.ScriptContext;
import javax.script.ScriptEngine;
import javax.script.ScriptEngineManager;
import javax.script.ScriptException;

// ...

ScriptEngineManager m = new ScriptEngineManager();
ScriptEngine rubyEngine = m.getEngineByName("jruby");
ScriptContext context = rubyEngine.getContext();

context.setAttribute("label", new Integer(4), ScriptContext.ENGINE_SCOPE);

try {
    rubyEngine.eval("puts 2 + $label", context);
} catch (ScriptException e) {
    e.printStackTrace();
}
```

Calling JRuby from Java: BSF

```java
import org.jruby.Ruby.*;
import org.jruby.*;
import org.jruby.javasupport.bsf.*;
import org.apache.bsf.BSFException;
import org.apache.bsf.BSFManager;
{...}
JLabel mylabel = new JLabel();
BSFManager.registerScriptingEngine("ruby",
                                   "org.jruby.javasupport.bsf.JRubyEngine",
                                   new String[] { "rb" });

BSFManager manager = new BSFManager();

/* Import an object using declareBean then you can access it in JRuby with $<name> */

manager.declareBean("label", mylabel, JFrame.class);
manager.exec("ruby", "(java)", 1, 1, "$label.setText(\"This is a test.\")");
```

Metaprogramming with Ruby

Structures

```ruby
Person = Struct.new "Person", :first_name, :last_name
p1 = Person.new
p1.last_name = "Doe"
p1.first_name = "John"
p1 # => #<struct Struct::Person first_name="John", last_name="Doe">
p2 = Person.new("Jane", "Doe")
p2 # => #<struct Struct::Person first_name="Jane", last_name="Doe">
```

Creating Objects and Classes by Name

```
s = Kernel.const_get('String').new "Teststring" # => "Teststring"
s.class # => String

Test = Class.new # => Test
Test.class_eval do
  def test1
    "test1"
  end
end

Test.new.test1 # => "test1"

Test.class_eval do
  define_method "test2" do
    "test2"
  end
end

Test.new.test2 # => "test2"
```

Individual Object Methods

```
t1 = Test.new
t2 = Test.new
t1.standard_method # => "standard_method; self: #<Test:0x16ee0>"
t2.standard_method # => "standard_method; self: #<Test:0x16e04>"

class << t1
  def object_method
    "object_method; self: #{self}"
  end
end

t1.object_method # => "object_method; self: #<Test:0x16ee0>"
t2.object_method # => NoMethodError: undefined method ‘object_method’
```

Classes & Constants

```
cls = Class.new
cls.class_eval do
  define_method :test_method do
    "test_method"
  end
end

cls.new.test_method # => "test_method"
cls # => #<Class:0x1b2b0>

SomeArbitraryConstant = cls
cls # => SomeArbitraryConstant
```

Runtime Definitions

```
class TestClass
  puts "before definition, self: #{self}"
  def my_instance_method
    puts "my_instance_method, self: #{self}"
  end
  puts "after definition, self: #{self}"
end


# >> before definition, self: TestClass

# >> after definition, self: TestClass

# >> my_instance_method, self: #<TestClass:0x19f00>
```

Runtime Definitions (2)

```ruby
class TestClass
  def self.my_class_method
    puts "my_class_method, self: #{self}"
  end

  my_class_method
end


# >> my_class_method, self: TestClass
```

Methods Adding Methods

```
class Meta
  def initialize(value)
    @value = value
  end

  def self.add_multiplier(factor)
      define_method "times#{factor}" do
        @value * factor
      end
  end

  add_multiplier 5
end



Meta.new(3).times5 # => 15
```

Methods Adding Methods (2)

```
class MultiplyTest
  include Multiplication

  def initialize(value)
    @value = value
  end

  add_multiplier 3
end
MultiplyTest.new(3).times3 # => 15
```

```
module Multiplication
  module ClassMethods
    def add_multiplier(factor)
      define_method "times#{factor}" do
        @value * factor
      end
    end
  end

  def self.included(clazz)
    clazz.extend(ClassMethods)
  end

end
```

(Re-)Opening Classes

```
def to_label(s)
  (s.split '_' ).map {|c| c.capitalize}.join ' '
end

to_label("LONG_UNREADBLE_CONSTANT") # => "Long Unreadble Constant"
to_label("unwieldy_name") # => "Unwieldy Name"


class String
  def to_label
    (self.split '_' ).map {|c| c.capitalize}.join ' '
  end
end

"LONG_UNREADBLE_CONSTANT".to_label # => "Long Unreadble Constant"
"unwieldy_name".to_label # => "Unwieldy Name"
```

(Re-)Opening Classes (2)

```
def array_shuffle!(array)
  0.upto(array.length-1) do |i|
    r = (rand * array.length).to_i
    array[i], array[r] = array[r], array[i]
  end
  array
end

array = %w(7 8 9 10 B D K A)
array_shuffle!(array) # => ["A", "D", "9", "7", "10", "8", "K", "B"]


class Array
  def shuffle!
    0.upto(length-1) do |i|
      r = (rand * length).to_i
      self[i], self[r] = self[r], self[i]
    end
    self
  end
end

array.shuffle! # => ["9", "B", "K", "A", "8", "10", "7", "D"]
```

method_missing

```
class Recorder
  def method_missing(name, *args)
    @calls ||= []
    @calls << { :name => name, :args => args}
  end

  def print_calls
    @calls.each do |call|
      puts "#{call[:name]}(#{call[:args].join(', ')})"
    end
  end
end

r = Recorder.new
r.first_call 1, 2, 3
r.second_call "Hello"
r.third_call :bumm
r.print_calls
# =>
# >> first_call(1, 2, 3)
# >> second_call(Hello)
# >> third_call(bumm)
```

Metaprogramming Examples

Rails ActiveRecord

```
class Project < ActiveRecord::Base
    belongs_to :portfolio
    has_one :project_manager
    has_many :milestones
    has_and_belongs_to_many :categories
end
```

Generated Methods

class Project < *ActiveRecord::Base*

belongs_to :portfolio	Project.portfolio Project.portfolio=(*portfolio*) Project.portfolio.nil?
has_one :project_manager	Project.project_manager, Project.project_manager=(*project_manager*) Project.project_manager.nil?
has_many :milestones	Project.milestones.empty? Project.milestones.size Project.milestones Project.milestones<<(*milestone*) Project.milestones.delete(*milestone*) Project.milestones.find(*milestone_id*) Project.milestones.find(:all, *options*) Project.milestones.build, Project.milestones.create
has_and_belongs_to_many :categories	Project.categories.empty? Project.categories.size Project.categories Project.categories<<(*category1*) Project.categories.delete(*category1*)

end

acts_as_state_machine

```
class Cat < ActiveRecord::Base
   acts_as_state_machine :initial => :sheltered, :column => 'status'
   state :sheltered #Initial state - Cat is at the shelter being cared for
   state :incare # Cat is with a shelter appointed carer
   state :returned # Owner located and cat returned
   state :housed # New owner is found for cat
   event :shelter do
     transitions :to => :sheltered, :from => :incare
   end
   event :care do
     transitions :to => :incare, :from => :sheltered
   end
   event :return do
     transitions :to => :returned, :from => :sheltered
     transitions :to => :returned, :from => :incare
   end
   event :house do
     transitions :to => :housed, :from => :sheltered
     transitions :to => :housed, :from => :incare
   end
end
```

Atom with XML Builder

see: http://intertwingly.net/stories/2005/09/21/app/views/blog/atom.rxml

```
xml.instruct! 'xml-stylesheet', :href=>'/stylesheets/atom.css', :type=>'text/css'
xml.feed :xmlns=>'http://www.w3.org/2005/Atom' do
  xml.div :xmlns=>'http://www.w3.org/1999/xhtml', :class=>'info' do
   xml << <<-EOF
     This is an Atom formatted XML site feed.
     It is intended to be viewed in a Newsreader or syndicated to another site.
     Please visit <a href="http://www.atomenabled.org/">atomenabled.org</a> for more info.
  EOF
  end
  xml.title   'Sam Ruby'
  xml.link    :rel=>'self',
    :href=>url_for(:only_path=>false, :action=>'posts', :path=>['index.atom'])
  xml.link    :href=>url_for(:action=>'posts', :path=>nil)
  xml.id      :href=>url_for(:only_path=>false, :action=>'posts', :path=>nil)
  xml.updated Time.now.iso8601
  xml.author  { xml.name 'Sam Ruby' }
  @entries.unshift @parent if @parent
  @entries.each do |entry|
    xml.entry do
      xml.title   entry.title
      xml.link    :href=>url_for(entry.by_date)
      xml.id      entry.atomid
      xml.updated entry.updated.iso8601
      xml.author  { xml.name entry.author.name } if entry.author
      xml.summary do
        xml.div :xmlns=>'http://www.w3.org/1999/xhtml' do
          xml << entry.summary
        end
      end if entry.summary
      xml.content do
        xml.div :xmlns=>'http://www.w3.org/1999/xhtml' do
          xml << entry.content
        end
      end
    end
  end
end
```

Resources

- John Hume: A Ruby Metaprograming Introduction
  http://practicalruby.blogspot.com/2007/02/ruby-metaprogramming-introduction.html
- Ola Bini: Ruby Metaprogramming Techniques
  http://ola-bini.blogspot.com/2006/09/ruby-metaprogramming-techniques.html
- _why: Seeing Metaclasses Clearly
  http://whytheluckystiff.net/articles/seeingMetaclassesClearly.html

Q&A

Stefan Tilkov

stefan.tilkov@innoq.com

http://www.innoq.com/blog/st/

innoQ

Architectural Consulting

SOA	WS-*	REST
MDA	MDSD	MDE
J(2)EE	RoR	.NET

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone +49 21 02 77 162-100
info@innoq.com · www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Phone +41 41 743 01 11

http://www.innoq.com